Adaptive Tetrahedral Grid Refinement and Coarsening in Message-Passing Environments

Jackie P. Hallberg[1]
Alan K. Stagg[2]
Joseph H. Schmidt[3]

[1] Engineer Research and Development Center
Coastal and Hydraulic Laboratory
3909 Halls Ferry Road
Vicksburg, MS 39180
pettway@juanita.wes.army.mil

[2] Los Alamos National Laboratory
Applied Physics Division
P.O. Box 1663, MS F645
Los Alamos, NM 87545
stagg@lanl.gov

[3] 2420 Wanda Way
Reston, VA 20191
roig.and.schmidt@erols.com

Abstract

A grid refinement and coarsening scheme has been developed for tetrahedral and triangular grid-based calculations in message-passing environments. The element adaption scheme is based on an edge bisection of elements marked for refinement by an appropriate error indicator. Hash-table/linked-list data structures are used to store nodal and element information. The grid along inter-processor boundaries is refined and coarsened consistently with the update of these data structures via MPI calls. The parallel adaption scheme has been applied to the solution of a transient, three-dimensional, nonlinear, groundwater flow problem. Timings indicate efficiency of the grid refinement process relative to the flow solver calculations.

Introduction

Adaptive grid methods based on point insertion and removal have been popular for a number of years for achieving greater solution accuracy with relative cost efficiency. This approach is used to resolve dynamic and fine-grid scale phenomena. The advantage is that fine-scale flow features can be captured without uniformly gridding the domain with a costly, fine grid. However, issues related to implementing such schemes on parallel systems have only recently been addressed, and much work is needed to identify the best approaches.

We present our work in this area with application to finite element modeling in message-passing environments. Our grid refinement strategy based on point insertion has been described elsewhere [1]. In this paper we extend this refinement capability to enable local grid coarsening and describe this efficient approach for irregular tetrahedral and triangular grids. Data structures have been selected to simplify implementation and coding complexity as much as possible for refinement, coarsening, and load balancing components. This software is being utilized in the Department of Defense code ADH (Adaptive Hydrology) developed at the U.S. Army Engineer Research and Development Center. ADH is a modular, parallel, finite element code designed to simulate flow in groundwater and two- and three-dimensional surface water [2].

Serial Element Adaption Scheme

Given an initial grid, the model subdivides grid elements according to an explicit error indicator to achieve the desired resolution in regions of need. Elements can be merged to increase efficiency where increased resolution is no longer required. This process is illustrated in Figure 1, where a new node is added to an edge, creating two new tetrahedra. The new elements may be merged to recover the original element by removing the inserted node.

The parallel grid adaption scheme developed here is based on the serial, geometric, edge-splitting algorithm of Liu and Joe [3]. Element edges are selected for subdivision based on a modified longest-edge bisection approach in which the oldest edge is flagged for bisection. Here the age of an edge indicates the level of refinement during which the edge was created. If the edges have the same age (as would be the case for the initial unrefined grid), then the longest edge is flagged for bisection. A grid closure step involving additional edge bisection is used to eliminate hanging nodes generated during the refinement step. This generation of a conforming grid is required before the coarsening phase can begin.

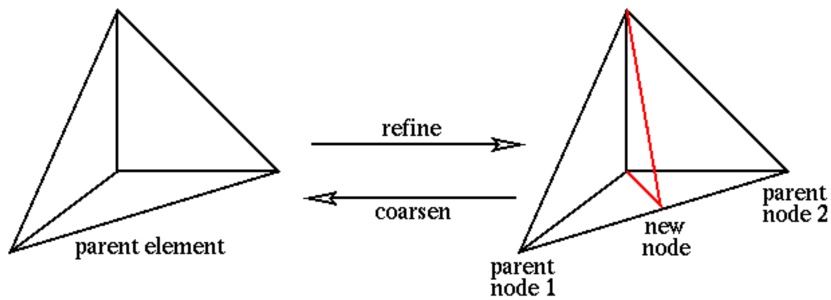

Figure 1. Tetrahedral Grid Adaption Based on Edge Bisection

The first step in the coarsening phase is to mark elements as potential candidates to be coarsened according to an appropriate error indicator. Some of these elements may not be allowed to merge with neighbor elements, depending on the error associated with the neighbor elements. Also, a refined grid can be coarsened only to the extent of the original grid. In other words, an unrefined or original element cannot be merged with another element.

In this adaption scheme, the coarsening process occurs precisely in reverse order of the refinement process at the element level. That is, the new node created on an edge during element refinement is the same node removed to merge the two elements during coarsening. To identify the last node added within an element, element node levels are maintained during the refinement process. Higher node levels indicate newer nodes. For two elements to merge and restore the parent element, nodal data for the nodes adjacent to the one being removed is required. This node-adjacency information is set during refinement by storing the nodes of the edge being split. When two elements agree to merge and remove the newest node, then the node-adjacency information for that node is retrieved, and the first of the merging elements to be processed restores the parent element. The following pseudo-code describes the basic element refinement and coarsening scheme.

Grid adaption pseudo-code

```
loop over elements
        refine element via edge bisection if its error > tolerance
conforming_grid = false
do while conforming_grid == false {
        conforming_grid = true
        loop over elements
                if element has an edge with a newly inserted node{
                        refine element
```

```
                    conforming_grid = false
            }
}
initialize node removal flags to YES
loop over elements
        set node removal flags to NO for old node and for all element nodes if
                error is high
loop over elements
        if any nodes are flagged to be removed, then merge the elements
```

Parallel Implementation of Element Adaption Scheme

Parallel implementation of local grid refinement schemes, like the edge bisection scheme above, presents a number of challenges. First, in the standard approach where the grid is partitioned and geographic subregions are assigned to processors, these subregions must be refined and coarsened consistently along processor boundaries. Also, closure requirements may force refinement to spread to a processor that has no elements marked for refinement by the error indicator. Finally, the local adaption process will likely lead to load imbalance among the processors, and nodes and elements must be transferred among processors during dynamic load balancing so that processing efficiency is maintained.

In our approach grid partitioning is accomplished by assigning element nodes uniquely to processors. Processors owning the element nodes share elements along processor boundaries. Nodal information for these elements is communicated among processors using MPI, and each processor stores complete data for its shared elements [4]. The ability of the code to run on several parallel platforms and maintain portability was a major concern during development.

Data Structures

Data structures were selected to simplify the parallel implementation of the adaption scheme and to facilitate the coupling of the refinement, coarsening, and load balancing components. During early work, we realized that common techniques like the use of tree structures for refinement could adversely impact other adaption components such as load balancing. In this case, the use of graph partitioners and the resulting grid point movement between processors requires splitting refinement trees between processors. To avoid the difficulties associated with splitting trees between processors and subsequent grid coarsening, we chose to use hash-table/linked-list structures [5]. Such structures are naturally suited for grid adaption since they are dynamic in nature and facilitate node and element searches. These structures handle all grid refinement, coarsening, and load balancing needs without complicating the implementation of any single component.

Hash tables are used to store nodes and element edges. Each entry in the node hash table consists of a local node number relative to the owning processor and corresponding node identifier in the global grid. Each entry in the edge hash table consists of the two local node numbers that define the edge, an integer edge rank based on comparative lengths of the edges, and an integer that stores the new node number if a node is inserted on the edge. Prior to refinement, the node and edge hash tables are allocated and filled; this memory is freed once the refinement process, including closure, is complete. Similarly, the element hash tables are allocated and filled, and this memory is freed once the coarsening process is complete. The nodes and elements are renumbered after each refinement and coarsening phase, and load balancing is performed after a complete adaption cycle (refinement, closure, renumbering, coarsening, and renumbering) as necessary.

Grid Consistency among Processors

The grid refinement scheme presented here is primarily a local process and thus is amenable to parallel processing. The principal requirement in a parallel environment is that processors periodically communicate to maintain grid consistency along the inter-processor boundaries. In the serial case, the new node number on the edge is stored in the edge hash table, and the adjacent element checks for the presence of a new node in the hash table to see if refinement for closure is required. In our parallel approach, an edge that spans two processors will appear in each of these processors' hash tables, and a protocol must be established to maintain consistency of the edge hash tables between processors. To support this communication, edge communication lists are constructed which provide a mapping between these duplicated edge storage locations. For each such edge, one of the processors sharing the edge is assigned ownership of it.

An example is illustrated in Figure 2 where three elements are distributed over two processors as indicated by the shaded background. The two processors share the center and right elements. In the first step, processor P0 splits the left element because of high error. Next P0 splits the original center element in the closure phase because that element now has a new node on one of its edges. Note that the center element's longest edge is bisected rather than the edge with the new node. Following this second step, P0 communicates the new node number on the shared edge to P1 using the edge communication lists. This communication provides the necessary data for P1 to refine its copies of the center and right elements so that the shared grid region is identical on both processors. In this example grid refinement has spread from P0 to P1 even though P1 did not have any elements marked for refinement by the error indicator.

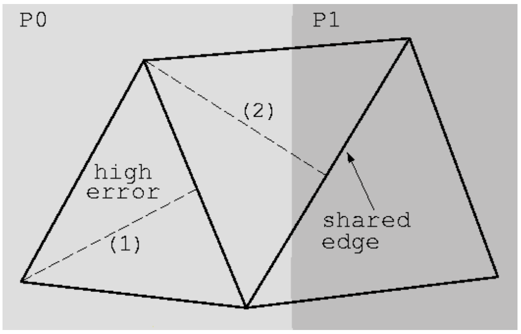

Figure 2. Edge Bisected by Owning Processor

Edge Ranking

After constructing the edge communication lists, edges are ranked based on their length so that they are uniquely and consistently identified throughout the global grid for the refinement phase. Integer rankings are utilized rather than using computed edge lengths so that processors are easily able to make consistent edge bisection decisions when multiple edges in an element are the same length.

Following a parallel, odd-even transposition sort, global ranks are returned to processors owning the edges, and these processors then store the ranks in their edge hash tables. These processors then communicate the ranks to the processors sharing the edges using the edge communication lists that have been constructed. The receiving processors finally store the ranks in their edge hash tables.

Element Refinement

After elements have been selected for refinement based on the error indicator, edges are selected for bisection based on their age and rank within the element. To refine an element, the oldest edge (or longest edge in a tie) in the element is bisected. To determine if another processor has already bisected the edge, the new node entry in the edge structure is inspected for that edge. If the edge has not been bisected, a new node is created for the edge, and the hash table is adjusted locally. Two new elements are created with the bisection of an edge, and the element Jacobians and other data are established for these new elements.

The new node entries for the edges in these new elements are reinitialized to indicate that new nodes are not present.

Closure

After elements have been refined based on the error indicator, further refinement might be required to obtain a closed grid (no hanging nodes). In the serial case, each element is checked for edges with new nodes via the edge hash table. If any element has an edge with a new node, that element is marked for refinement according to the established rules. The refinement process continues iteratively until a closed grid is obtained.

In a parallel environment this procedure is complicated by the fact that shared edges may be bisected by only one of the processors spanned by the edge. To maintain consistency of the edge hash tables, processors owning shared edges communicate new node information to processors sharing the edges. If a message indicates that an edge has a new node, then the receiving processor creates a new node for the edge and updates its hash table. Similarly, processors may bisect edges they do not own. To handle this situation, the edge communication lists are utilized in reverse order (the send list becomes a receive list, and vice versa) so that processors owning edges that are shared can update their hash tables if other processors bisect them. After this communication, the elements with new nodes on edges are refined, and the process is repeated until the grid is closed.

Parallel Coarsening

The parallel coarsening process begins with the communication of node removal flags (set by the error indicator) across processor boundaries. Node adjacency data is also communicated to allow for proper merging of elements across these boundaries. With this information each processor is able to merge its marked elements concurrently and independently. Each new, merged element is created only once using hash table lookups to avoid duplicate element creation.

Following element merging, the node levels for the elements created along the inter-processor boundaries must be updated to enable further coarsening of these elements if needed later. These node levels are obtained as follows. During the merging process, new elements created along the inter-processor boundaries are marked for outgoing node-level communication by storing their numbers in linked lists according to the identification numbers of the receiving processors. At this point, processors have only determined the destinations and message sizes for outbound messages, and the number of inbound messages is unknown. Global communication of the destination processor numbers is utilized so that each processor can calculate the number of messages it will receive. Each processor then sends its messages and probes for each incoming message to obtain the sending processor identification number and message size. With this information each processor is able to receive each inbound message and then set the element node levels for the new inter-processor boundary elements.

Cleanup

Following node level communication, the remaining steps in the coarsening process involve bookkeeping updates for the new grid. First, adjacent node data is set for the new grid, and unused nodes and elements are re-initialized. The nodes and elements are renumbered so that unused items are placed at the end of the lists in memory. Given the new grid numbering, the communication lists for exchanges between processors are updated. Finally, the global node identifiers and adjacent node data are updated using the new communication lists. At this point, the grid is ready for subsequent refinement and coarsening. Also, the grid may be repartitioned dynamically among the processors if desired.

Groundwater Application

The capabilities of the parallel grid refinement scheme have been investigated for the solution of a draining heterogeneous column. In this problem a column is filled with a mixture of clay, silt, and sand. The column consist primarily of sand with a clay lens near the bottom and silt lenses in several places throughout the column. Initially, the column is completely saturated with water, and then the water is allowed to drain from the bottom of the column. The grid is allowed to refine and coarsen locally as dictated by the explicit error indicator, and dynamic load balancing is used to improve processor efficiency.

A snapshot of the adaptively refined grid for the heterogeneous column is illustrated in Figure 3. The area shaded black represents the clay material, while the sand and silt are represented by the gray and white regions, respectively. Grid refinement is visible at the sand/silt interface in the lowest points of the sand. Water travels through the sand at a faster rate than through the silt and large pressure gradients will develop at the material interface. The refinement shows the points of local flow concentration.

Preliminary timings for the heterogeneous column described above were obtained on 1-8 processors of a Silicon Graphics Origin 2000. During the refinement step, the number of nodes was increased by 35%, and the total time spent in grid refinement was an order of magnitude less than the time spent in the flow solver. The time spent in grid closure, including communication, was less than half of the total time in grid refinement for these cases. Though preliminary, these timings indicate efficiency of the grid refinement process relative to the flow solver.

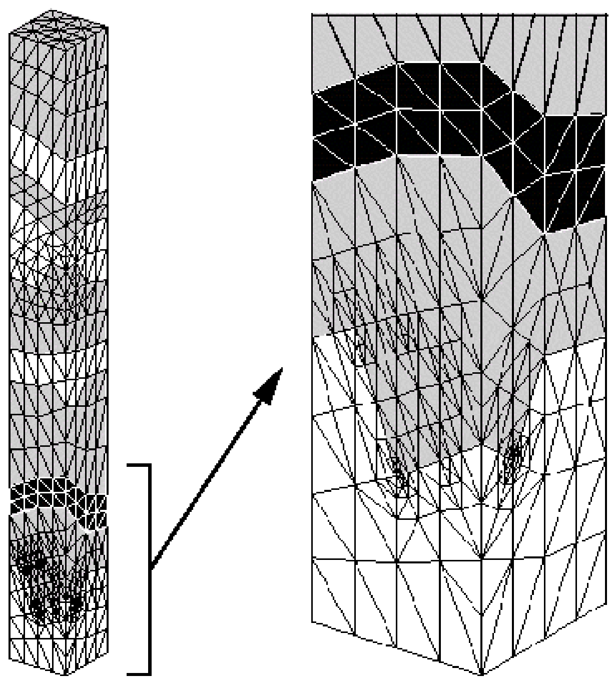

Figure 3. Adaptively Refined Grid for Heterogeneous Column

Conclusion

A parallel refinement scheme has been developed for tetrahedral and triangular grids. The refinement scheme and data structures described here have been developed to facilitate the parallel implementation of both grid refinement and coarsening. The refinement and coarsening schemes are based on communicating a minimum set of data and reconstructing information locally without the use of tree structures. The goal with this approach is a balanced design between refinement, coarsening, and load balancing in terms of efficiency and ease of implementation. Preliminary application of the adaptive grid scheme to an unsteady groundwater flow problem has demonstrated the capability and efficiency of the method.

Acknowledgment

Funding for this project was provided by the Department of Defense High Performance Computing Modernization Office. Permission to publish this paper was granted by the Chief of Engineers and by Los Alamos National Laboratory.

References

[1] Stagg, A.K., Hallberg, J.P., and Schmidt, J.H., "A Parallel, Adaptive Refinement Scheme for Tetrahedral and Triangular Grids", International Parallel and Distributed Processing Symposium Workshop Proceedings, Springer-Verlag Lecture Notes in Computer Science, Cancun, Mexico, May 2000.

[2] Jenkins, E.W., Berger, R.C., Hallberg, J.P., Howington, S.E., Kelley, C.T., Schmidt, J.H., Stagg, A.K., and Tocci, M.D., "Newton-Krylov-Scharz Methods for Richards' Equation", submitted to the *SIAM Journal on Scientific Computing*, October 1999.

[3] Liu, A. and Joe, B., "Quality Local Refinement of Tetrahedral Meshes Based on Bisection", *SIAM Journal on Scientific Computing*, vol. 16, no. 6, pp. 269-1291, November 1995.

[4] Snir, M., Otto, S., Huss-Lederman, S., Walker, D., and Dongarra, J., *MPI-The Complete Reference, Volume 1, The MPI Core*, The MIT Press, Cambridge, Massachusetts, 1998.

[5] Cormen, T., Leiserson, C., and Rivest, R., *Introduction to Algorithms*, The MIT Press, Cambridge, Massachusetts, 1990.